C++ DATA TYPES

While doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data type like character, wide character, integer, floating point, double floating point, boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Primitive Built-in Types:

C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table list down seven basic C++ data types:

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Several of the basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

The following table shows the variable type, how much memory it takes to store the value memory, and what is maximum and minimum vaue which can be stored in such type of variables.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127

int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

The sizes of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example which will produce correct size of various data type on your cmputer.

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;
   return 0;
}
```

This example uses **endl** which inserts a new-line character after every line and << operator is being used to pass multiple values out to the screen. We are also using **sizeof()** function to get size of various data types.

When the above code is compiled and executed, it produces following result which can vary from machine to machine:

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

typedef Declarations:

You can create a new name for an existing type using **typedef**. Following is the simple syntax to define a new type using typedef:

```
typedef type newname;
```

For example, the following tells the compiler that feet is another name for int:

```
typedef int feet;
```

Now, the following declaration is perfectly legal and creates an integer variable called distance:

```
feet distance;
```

Enumerated Types:

An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration.

To create an enumeration requires the use of the keyword **enum**. The general form of an enumeration type is:

```
enum enum-name { list of names } var-list;
```

Here, The enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called colors and the variable c of type color. Finally, c is assigned the value "blue".

```
enum color { red, green, blue } c;
c = blue;
```

By default, the value of the first name is 0, the second name has the value 1, the third has the value 2, and so on. But you can give a name a specific value by adding an initializer. For example, in the following enumeration, **green** will have the value 5.

```
enum color { red, green=5, blue };
```

Here **blue** will have a value of 6 because each name will be one greater than the one that precedes it.